Université Libre de Bruxelles

Vrije Universiteit Brussel

Machine Learning and Big Data Processing

ELEC-Y-591 - 010134

Matrix Completion Report

Implemented in Python 3.11

Students :
Al Khouri Khalil (IrPH)
Misselyn Alexis (IrEL)
Niedercorn Eliot (IrPH)
Tomson Eliot (IrPH)

Professors :
Munteanu Adrian
Deligiannis Nikolaos

Assistants :
Royen Remco
Yuqing Yang

Academic Year 2022-2023

Contents

1 Introduction

Among the many applications of neural networks, there is one that we are very often confronted to in our everyday life : recommendation algorithms. Whether it is on YouTube, Facebook or Netflix, most of what is suggested to us is predetermined by a neural network that will choose what content to show us in order to satisfy us based on our tastes. These recommendations are done by comparing the content we liked to what content other users that have watched similar content liked.

This process can be mathematically represented as a partially filled matrix, where observed elements correspond to the opinion of the users for a given content, and unknown elements represent the content for which we seek to predict the user's preferences. The task of inferring values for these unobserved elements is known as **Matrix Completion**.

This project is focused on the results obtained with different architectures of autoencoder-based matrix completion algorithms. First, important principles of neural network techniques for matrix completion will be explained in a theoretical section. Then the results obtained with a classical autoencoder as well as deep autoencoder and hybrid autoencoder will be compared. Finally, we will compare what we obtained with state of the art papers on these methods.

2 Objectives

The proposed project aims to undertake a comprehensive study to gain a deep understanding of the matrix completion problem and explore the current state of the art. By conducting a literature review, the project will identify and analyze various autoencoder-based approaches proposed in research papers to tackle this problem.

The primary focus of the project, however, lies in implementing a neural network-based approach to address the matrix completion problem. This implementation will serve as a basis for comparison against the results obtained by other existing approaches. Through this comparative analysis, the project seeks to evaluate the performance and effectiveness of the neural network-based solution in relation to the methods proposed in the reviewed papers.

3 State of the Art

In this section, we delve into the methods for matrix completion, with a focus on autoencoders. We thoroughly explore the intricacies of autoencoder architecture, its application in matrix completion, and the challenges it tackles. Later, we define a denoising autoencoder and discuss how it contributes to matrix completion.

3.1 Matrix Completion

Matrix completion is the task of inferring missing values in a matrix that is only partially observed. This task is commonly encountered in scenarios where we have user-item matrices, with items represented as columns and users as rows. Each entry in the matrix corresponds to the rating a user has given to a specific item. In such scenario, most of the users do not provide ratings for the majority of items, resulting in a matrix with mostly missing values.

By developing an accurate matrix completion algorithm, one can construct a recommendation system that would advise users on which items they are more likely to be interested in.

The traditional approach to performing matrix completion is to factorize the partial matrix into particular matrices, whose multiplication reconstructs the matrix. This approach commonly employs methods based on **nuclear-norm minimization** or **truncated nuclear norm minimization**. [2]

An alternative approach utilizes neural networks such as **Restricted Boltzmann machines** [6] or **Denoising autoencoder** [2, 8] to fill these matrices. The denoising autoencoder approach is the one that we consider in this project.

3.2 Autoencoder (AE)

An **autoencoder** is a particular neural network architecture in which the input of the neural network is encoded into hidden layers through a series of transformations, and then decoded to retrieve the original inputs. In the context of matrix completion, the autoencoder operates on a matrix X with dimensions $(n \times m)$ and learns to perform a function that maps the input matrix X to an approximation of itself, denoted as $\hat{X}$. The elements of the input matrix are represented as x_{ij}, and the elements of the approximation matrix are denoted as $\hat{x}_{ij}$.

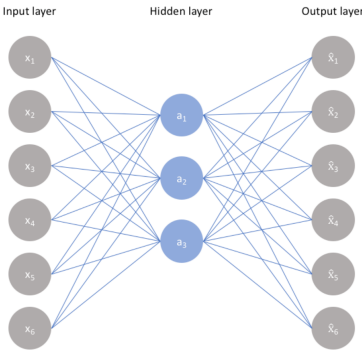

Figure 1: Representation of the architecture of a single hidden layer autoencoder.[1]

For matrix completion, the objective of the autoencoder is to learn the underlying patterns in the data and fill in the missing values. More specifically, in the user-item context, we want the autoencoder to learn the overall patterns between users and items. There are multiples way to achieve this. As mentioned in [8], one approach is the **user-based** approach, where the autoencoder takes the ratings provided by a single user i as input represented by the corresponding row $x_{i\_}$ and then attempts to construct its ratings for every item $\hat{x}_{i\_}$. This approach corresponds to guess the taste of the user based on item he likes. Another approach is the **item-based** approach, where the autoencoder takes the ratings provided for a single item j, corresponding to $x_{\_j}$, and then constructs its rating for every user $\hat{x}_{\_j}$. This corresponds to guessing the taste of a user based on finding users who are similar in taste. In [8], their results suggest that the user-based approach is slightly more effective, which is why we have chosen to incorporate it into our project.

[1] Image taken from `https://www.jeremyjordan.me/autoencoders/`.

As discussed in [8], there are two notable challenges for matrix completion using an autoencoder. Firstly, the neural network must be able to handle undefined values as inputs. Secondly, in addition to its primary purpose of reconstructing the initial input, the autoencoder must also predict the missing ratings. To address the first challenge, we set all the undefined values to zero and we discard these values in the backpropagation process, where the model weights are updated for optimization. To tackle the second challenge, we use a **denoising autoencoder** as presented in [9]. It consists in encouraging the model to learn underlying patterns rather than solely focusing on reconstructing the observed values. This is done by introducing corruption to the input data during the training process. This corruption randomly alters some of the observed entries in the input matrix. The autoencoder is then forced to learn and reconstruct the missing entries based on the remaining observed entries, with the corruption randomly evolving at each iteration of the training.

To incorporate the corruption in the autoencoder, a new loss function can be used, which consists in the sum of two components:

- The mean squared error (MSE) of the input and corrupted input vectors for the elements that are both observed (which is denoted by : $\in K(x)$) and corrupted (denoted as : $\in C(\tilde{x})$ where $\tilde{x}$ is the corrupted input), this corresponds to quantifying the error on the corrupted values.
- The MSE of the input and the corrupted input vectors, for the elements that are both observed and non-corrupted, corresponds to quantifying reconstruction quality.

By summing these two losses, we create a comprehensive loss function that addresses both aspects of the denoising autoencoder's objectives. The balance between the two losses can be controlled by multiplying them with hyperparameters α and β that can be used for fine-tuning. The loss function takes this form :

$$L_{\alpha,\beta}(x,\tilde{x}) = \alpha(\sum_{j\in K(x)\cap C(\tilde{x})}(nn(\tilde{x})_j - x_j)^2) + \beta(\sum_{j\in K(x)\setminus C(\tilde{x})}(nn(\tilde{x})_j - x_j)^2) + \lambda||W||^2 \tag{1}$$

Where W is the weight vector, λ is the weight decay and $nn(\tilde{x})$ is the output of the neural network given x as an input. The architecture corresponding to the above discussion is represented in Figure 2.

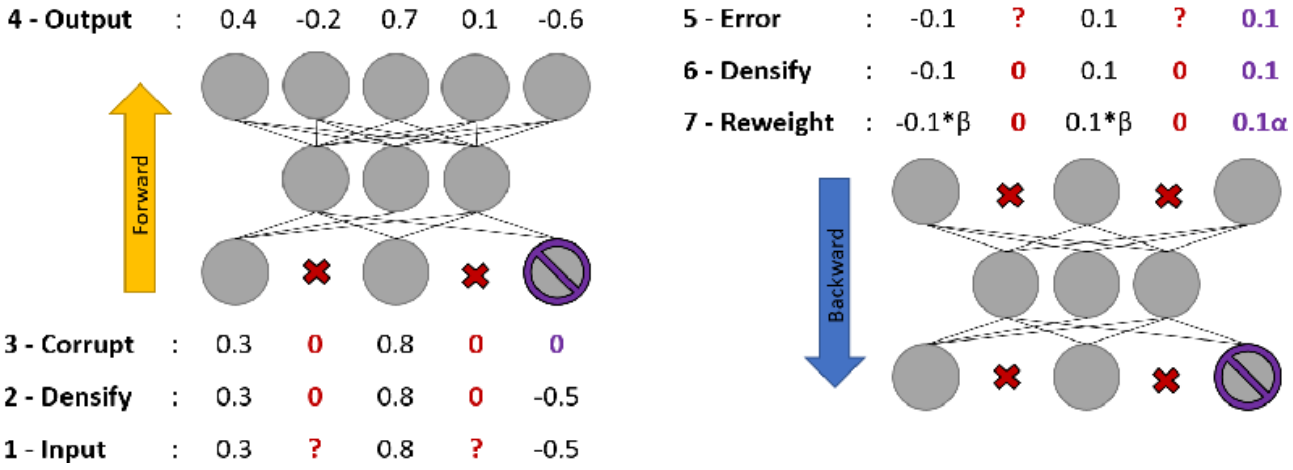

Figure 2: Operation representation of the **denoising autoencoder** as shown in Figure 1 of [8].

3.3 Stacked Autoencoders (SAE)

The idea is the same as a classical autoencoder while introducing a modified architecture known as stacked autoencoders. The variation involves putting in multiple pairs of encoders and decoders, distinguishing it from the classical autoencoder. The inspiration for this concept came from [5]. The design simply has multiple hidden layers instead of one like the classical autoencoders.

matrix that one want to complete

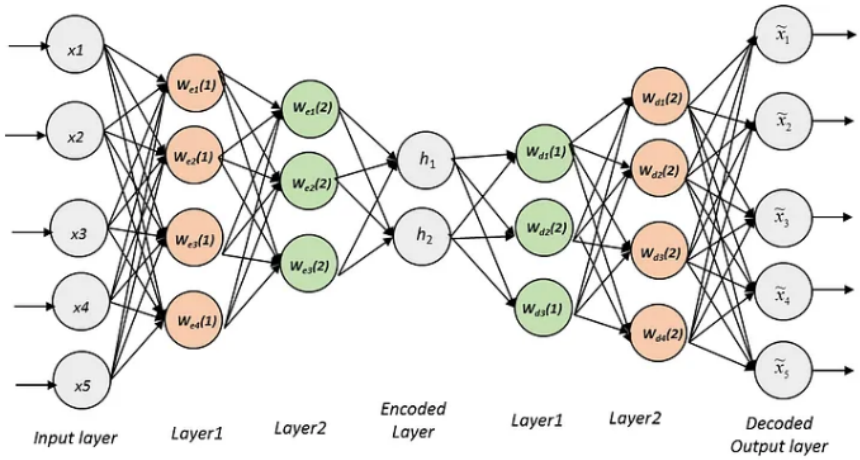

Figure 3: The proposed architecture from [3]

The patterns that could exist in a partially observed matrix are often non-linear; the articles [1, 4] actually proved that non-linear activation functions like ReLU (Rectified Linear Unit) have a better chance of uncovering the hidden patterns in the matrix than linear activation functions. And they proved that the SELU function had the best results. Another architecture

was proposed by [5] which had a linear and a non-linear implementation done in parallel, as in Figure (4).

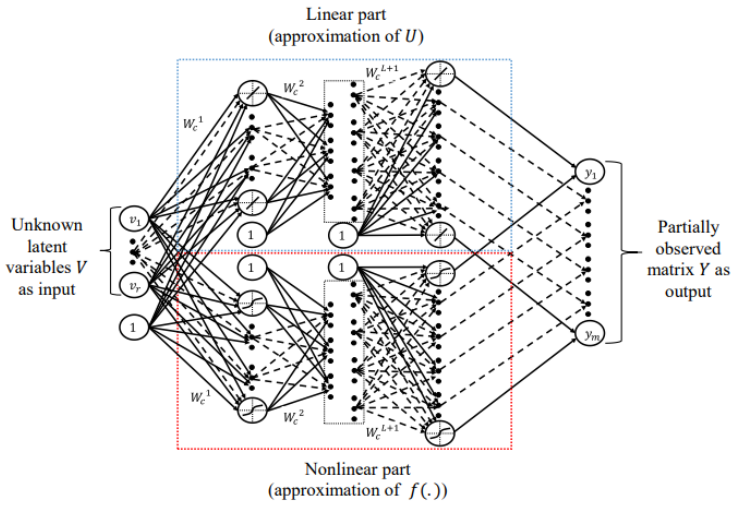

Figure 4: The proposed architecture from [5]

This architecture was not implemented, however, in this project because the simple non-linear ReLU stacked autoencoder have better results than the proposed architecture in figure(4).

3.4 Hybrid Autoencoder

There is an additional problem that arises when a new item is added to the set of items. As no one has rated it yet, the classical autoencoder approach won't be able to give it a rating, this is called the cold start problem. One approach to this problem involves incorporating additional information, as presented in [8]. This ancillary information also provides the model with the ability to comprehend supplementary patterns connecting users and items thus increasing the performance of the algorithm.

To illustrate this concept, let's consider a scenario where the items to be rated are movies, and the supplementary information consists of providing the genre of each movie. By analyzing user ratings and considering the genre information, the model can learn the user's preferred movie genres, enabling it to make more precise predictions for movies with unknown ratings, taking into account whether the genre aligns with the user's genre preferences or not.

A description of how this hybrid approach can be implemented is given in [8], to summarize it, the side information is injected as biases in every layer of the autoencoder. Although this project did not implement this approach, it presents a potential way to enhance its performance. However, it is important to note that the improvements for this implementation observed in Table 1 of [8] were relatively modest. But even if the improvements are small, the users and items with few ratings benefits more from these side information, which is very desirable when considering new items or new users.

4 Description of our approach

This section aims to describe the main ideas of our code.

4.1 Considered Datasets

In this project, we utilize two datasets: **Jester** and **MovieLens**. Both datasets are preprocessed to ensure compatibility with the autoencoder model. These datasets both have missing values, the general idea is to shape the matrices in order to keep only the relevant information, standardize the values of the inputs and create a mask corresponding to the known values in order to be able to select known and unknown values.

4.1.1 Jester

Jester[2] is a collection of datasets that comprises ratings of various jokes provided by users. Specifically, the dataset that we use from this collection is the one called "*jester_ dataset_ 1_ .zip*" which consists of 24,983 users rating 100 jokes. This dataset is given as a matrix of dimensions 24,983 X 101 where each row corresponds to a user and each column represents a joke. The matrix values are ratings, ranging from -10.00 to +10.00. Notably, each user has rated at least 36 jokes, resulting in approximately 72% of the ratings being available in this matrix. In this dataset, the first column gives the number of jokes rated by the corresponding user and the value "99" is used to denote a null or unrated joke.

First, we remove the first column of the dataset, as it does not provide insights into the users' preferences. Then we standardize the data to have scores between 0 and 1 for each entry with the following operation:

$$X_{ij} = \frac{X_{ij} + 10}{20} \qquad \forall i, j \tag{2}$$

4.1.2 MovieLens

MovieLens[3] is a collection of datasets that comprises ratings of movies provided by users. Specifically, the dataset that we use from this collection is the one called "MovieLens Latest Datasets "[4] which consist of 100,000 ratings from 600 users on 9000 movies. Each user has rated at least 20 movies, resulting in a data availability of approximately 6.3%. The matrix values are ratings, ranging from 0.5 to 5, so a standardization of the data is performed:

$$X_{ij} = \frac{X_{ij} - 0.5}{4.5} \qquad \forall i, j \tag{3}$$

4.2 Train-test separation

We choose a training proportion of 80%, this leaves a test set of 20% of the original dataset. The test set will ensure that there is no overfitting on the training set.

[2]https://goldberg.berkeley.edu/jester-data/
[3]https://grouplens.org/datasets/movielens/
[4]https://grouplens.org/datasets/movielens/latest/

4.3 Corruption

We will corrupt the matrix of each batch at each iteration. To do this, we create a mask that randomly selects certain values with a proportion p that we set at 0,3. This mask has the same dimension as the batch and is made of 0 and 1's. As such, 30% of the input values will be corrupted, thus set as 0.5 (the middle of the interval) in order for them to not be biased towards either of the extremes. We also derive from, this corruption mask, a mask of uncorrupted values in order to be able to isolate these values during the loss computation.

4.4 Learning

We use the loss as defined is section 3 by applying the criterion "MSELoss" included in Pytorch on both the observed corrupted matrix and the observed uncorrupted matrix, pondered by coefficients α and β respectively. We set these parameters as 0.7 and 0.3 as we want top put the accent on predicting non-existing values more than finding thos we already had.

As optimizer, we choose ADAM, which combines the advantages of both ADAGRAD (ADAptive GRAdient Descent) and RMSProp (Root Mean Square Propagation). The learning rate and the weight decay will be determined in section 5.1.

5 Experimental Results

The results section will be divided into parts; the first will be dedicated to the optimization of parameters (number of hidden features, batch-size, etc.), the second part will be the comparisons between the results obtained from our model and the results published in the state of the art.

5.1 Parameters optimization

The idea is to repeat the same test multiple times while changing a single parameter and try to find out which given value for the parameter is the best. It is important to keep in mind that the results obtained here can slightly vary considering the random nature of the training.

The following tests were done with the following parameters: 60 epochs, a learning rate of 0.0001, and no weight decay using a subset of the Jester dataset (5000x100).

5.1.1 Batch size

Loss in %

Batch size	AE 5	SAE 10 5 10
16	**4.77**	5.09
32	4.88	**4.66**
64	4.89	4.84
128	4.96	4.89

Table 1: Batch size optimization

AE 5 is the classical autoencoder with 5 features in the hidden layer, and SAE 10 5 10 is the stacked autoencoder with 3 hidden layers containing 10, 5, and 10 features, respectively. These values for the hidden layers are proposed in [2]. The best batch sizes appear to be 32 for the stacked autoencoders of layers 10–5–10 and 16 for the classical autoencoder.

5.1.2 Number of hidden features

The number of hidden features has to be smaller than the number of inputs; otherwise, the autoencoder could only memorize the values of the input matrix and replicate them without trying to understand the underlying patterns linking the values. It also has to have the right size in order to focus on capturing the most relevant relationships between the inputs and the outputs without missing too much information.

5.1.3 Weight decay

In this test, several regularization parameters will be tried on the Jester dataset. The chosen regularization method is weight decay, and the tested values are 0, 0.01, and 0.1.

Jester dataset: loss in %

λ	AE 5	SAE 10 5 10
no decay	**5.28**	**5.13**
0.01	5.51	5.46
0.1	9.85	7.11
1	34.84	35.12

Table 2: Weight decay parameter optimization for the Jester dataset

A weight-decay regularization of the small regularization parameter $\lambda = 0.01$ seems to give the best results for the Jester dataset. It is also important to note that the losses displayed in table (2) are the validation losses, which is the right thing to watch in the context of regularization. Based on the results of the test, it is safe to assume that the Jester dataset do not need regularization, and this could be understood by the fact that 72% of the entries are observables. Now the MovieLens dataset is considered (with only 6.3% of observed entries).

MovieLens dataset: loss in %

λ	AE 40	SAE 60 40 60
no decay	28.40	17.89
0.01	**10.61**	**9.13**
0.1	11.36	11.78
1	53.42	50.18

Table 3: Weight decay parameter optimization for the MovieLens dataset

The results are a bit worse than from the Jester dataset and that's because of the sparsity aspect of the MovieLens dataset.

5.2 Comparison with the state of the art

The objective of this section is to replicate the tests that were done in various papers with our model and compare the two results. Keep in mind that the tests were done in some setup that was replicated with our model and the displayed losses were not calculated in the same fashion. We used equation () to evaluate the loss

5.2.1 Comparison with *Deep learning based matrix completion*

This section is to display the comparison of the results between our model and one proposed by [2]. In this article, they propose two models: an autoencoder-based matrix completion (AEMC) and a deep learning-based matrix completion (DLMC) and here is the table comparing these two methods to our model (SAE).

NMSE in %

Dataset	AEMC	DLMC	SAE
Jester	15.74	15.53	**11.64**
MovieLens	18.52	18.21	10.21

Table 4: Comparison with [2]

5.2.2 Comparison with *Deep Learning Approach for Matrix Completion Using Manifold Learning*

The comparison in the section is between SAE and the model proposed by [5].

NMSE in %

Dataset	Article	SAE
Jester	**15.74**	16.56
MovieLens	**17.84**	18.77

Table 5: Comparison with [5]

The same tests that were done in the paper were repeated, and the SAE was not able to have the same efficiency as the model proposed in this paper.

5.3 Illustration of our autoencoder convergence

To illustrate the convergence of our autoencoder, we represent in Figure 5 the evolution of the losses for our train data and our validation data for our single layer autoencoder considering the Jester dataset with the fine-tuned parameters:

We can compare by looking at the result for our stacked autoencoder as shown in Figure 5

num_epochs	batch_size	num_hidden_neurons	learning_rate
120	128	12	0.0001
weight_decay	**alpha**	**beta**	**corruption_prob**
0.001	0.7	0.3	0.3

Table 6: Fine-tuned Parameters for our autoencoder on Jester Dataset

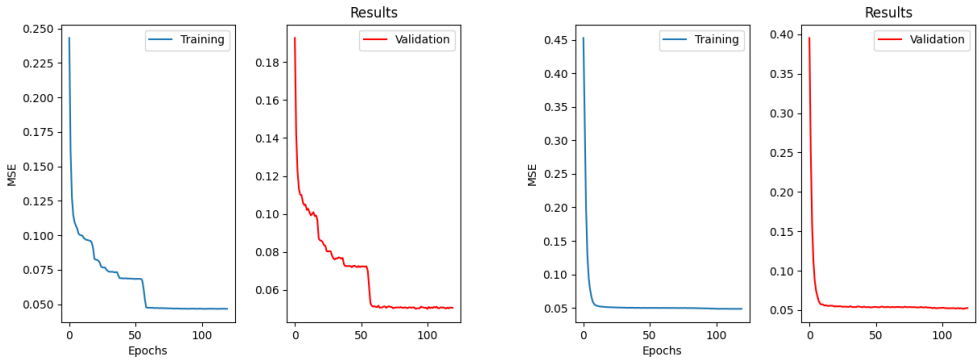

Figure 5: Convergence example for single layer AE (left) and stacked AE (right) on the Jester dataset

6 Conclusion

In this report, a neural network-based approach of matrix completion with autoencoders is presented. Indeed, three variations of autoencoder architecture are presented: the classical single hidden layer autoencoder, the stacked/deep autoencoder, and hybrid autoencoders.

The project then described the approach of the group, including the datasets used (Jester and MovieLens), train-test separation, corruption of input values, and the learning process. The objective was to optimize parameters such as batch size, number of hidden features, learning rate, and weight decay in order to have the best accuracy possible.

Also, a comparative analysis is drawn between the state-of-the-art and the results of the project to evaluate the performance of our neural network-based solution.

Overall, this project allowed us to have a deeper experience of more complex neural networks and exposed our group to the practical applications of neural networks such as the preparation of the dataset, parameter tuning, and the interpretation and comparison of the results of our implementations and optimizations, all in the context of matrix completion.

7 Appendix

7.1 Activation functions

Figure (6) shows several activation functions. In the context of matrix completing, the used functions are ReLU and SELU. The image was taken from [7].

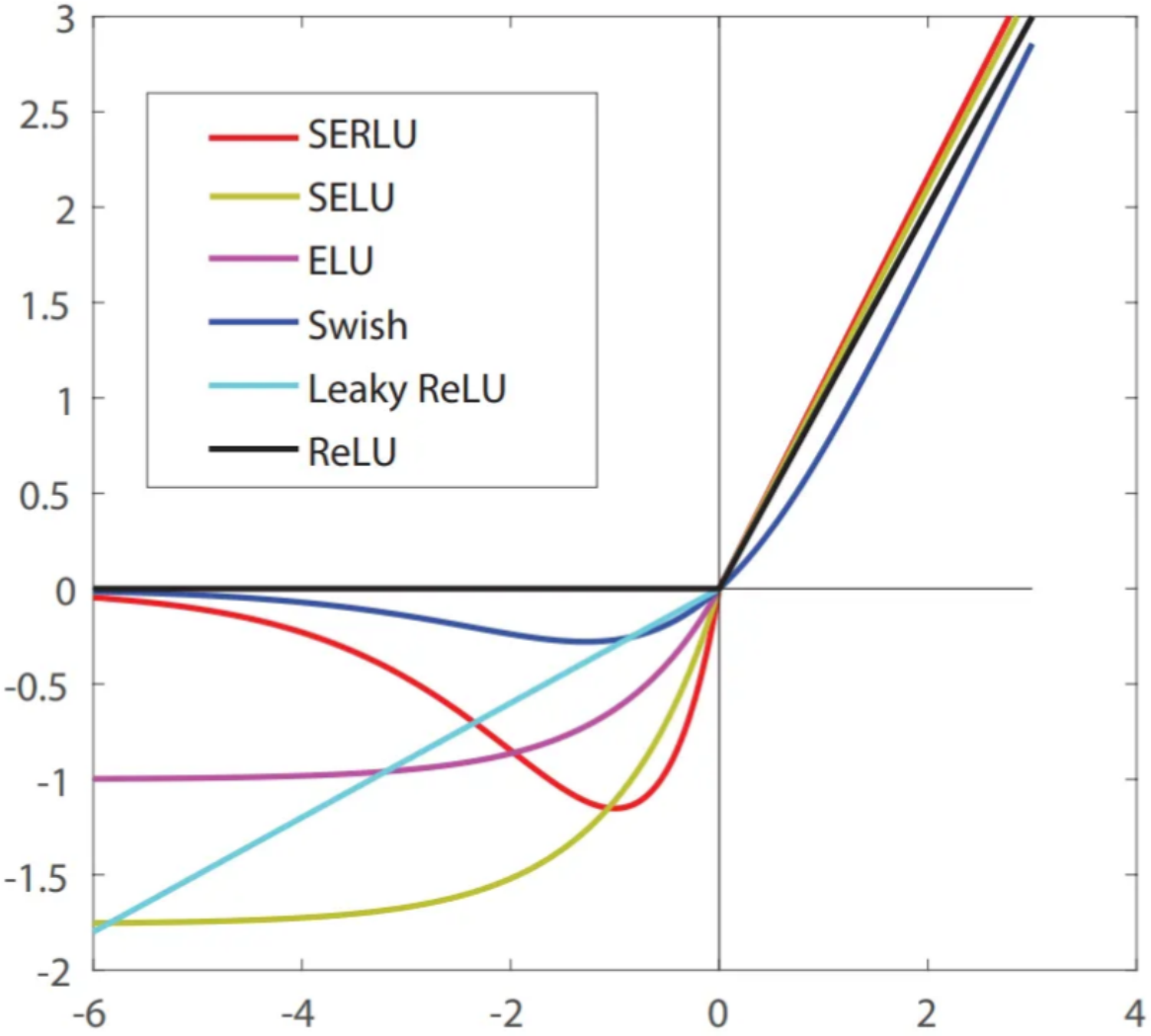

Figure 6: Several activation functions

References

[1] Omar Alhadlaq and Arjun Kunna. *Deep Learning-Based Collaborative Filtering*. 2018. URL: https://cs230.stanford.edu/files_winter_2018/projects/6937581.pdf.

[2] Jicong Fan and Tommy Chow. "Deep learning based matrix completion". In: *Neurocomputing* 266 (2017), pp. 540–549. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.05.074. URL: https://www.sciencedirect.com/science/article/pii/S0925231217309621.

[3] Venkata Krishna Jonnalagadda. *Sparse, Stacked and Variational Autoencoder*. en. 2018. URL: https://medium.com/@venkatakrishna.jonnalagadda/sparse-stacked-and-variational-autoencoder-efe5bfe73b64.

[4] Oleksii Kuchaiev and Boris Ginsburg. *Factorization tricks for LSTM networks*. arXiv:1703.10722 [cs, stat]. Feb. 2018. DOI: 10.48550/arXiv.1703.10722. URL: http://arxiv.org/abs/1703.10722.

[5] Saeid Mehrdad and Mohammad Hossein Kahaei. "Deep Learning Approach for Matrix Completion Using Manifold Learning". In: *Signal Processing* 188 (Nov. 2021). arXiv:2012.06063 [cs, stat], p. 108231. ISSN: 01651684. DOI: 10.1016/j.sigpro.2021.108231. URL: http://arxiv.org/abs/2012.06063.

[6] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. "Restricted Boltzmann machines for collaborative filtering". In: *ACM International Conference Proceeding Series* 227 (June 2007), pp. 791–798. DOI: 10.1145/1273496.1273596.

[7] Ygor Serpa. *A Comprehensive Guide on Activation Functions*. en. Oct. 2021. URL: https://towardsdatascience.com/a-comprehensive-guide-on-activation-functions-b45ed37a4fa5.

[8] Florian Strub, Romaric Gaudel, and Jérémie Mary. "Hybrid Recommender System Based on Autoencoders". In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. DLRS 2016. Association for Computing Machinery, 2016, pp. 11–16. ISBN: 9781450347952. DOI: 10.1145/2988450.2988456. URL: https://doi.org/10.1145/2988450.2988456.

[9] Pascal Vincent et al. "Extracting and Composing Robust Features with Denoising Autoencoders". In: ICML '08 (2008), pp. 1096–1103. DOI: 10.1145/1390156.1390294. URL: https://doi.org/10.1145/1390156.1390294.